Exploring Views for Goal-Oriented Requirements Comprehension

Lyrene Silva[1(✉)], Ana Moreira[2], João Araújo[2], Catarina Gralha[2], Miguel Goulão[2], and Vasco Amaral[2]

[1] Dimap/IMD, Federal University of Rio Grande do Norte, Natal, Brazil
lyrene@dimap.ufrn.br
[2] NOVA LINCS, DI, FCT, Universidade NOVA de Lisboa, Lisbon, Portugal
{amm,joao.araujo,mgoul,vma}@fct.unl.pt, acg.almeida@campus.fct.unl.pt

Abstract. Requirements documents and models need to be used by many stakeholders with different technological proficiency during software development. Each stakeholder may need to understand the entire (or simply part of the) requirements artifacts. To empower these stakeholders, views of the requirements should be configurable to their particular needs. This paper uses information visualization techniques to help in this process. It proposes different views aiming at highlighting information that is relevant for a particular stakeholder, helping him to query requirements artifacts. We offer three kinds of visualizations capturing language and domain elements, while providing a gradual model overview: the big picture view, the syntax-based view, and the concern-based view. We instantiate these views with *i** models and introduce an implementation prototype in the iStarLab tool.

Keywords: Requirements exploration · Visualization · Comprehension · Views

1 Introduction

Information exploration tasks, such as zooming, obtaining details-on-demand, filtering, extracting, relating, and overviewing [1] are basic tasks for information analysis. They have been used in several areas, including software engineering. For code exploration, for example, these tasks may help the maintenance software engineer to comprehend the structure and behavior of a program through the generation of multiple views [2]. Multiple views are also broadly employed in requirements modeling for very specific purposes. Usually, these views do not offer interactive features to allow stakeholders browsing the information according to their needs [3–6].

Requirements artifacts are usually described textually or graphically (e.g., with use case scenarios, or NFR graphs). These artifacts are often too large or too complex to be quickly understood or easily queried for information of interest by different stakeholders, including clients, domain experts, and software engineers. Therefore, how to navigate through requirements artifacts to get the relevant information? In other words, how can we explore (i.e., examine or study) requirements that have been elicited and documented (often by other people and/or not recently) to accomplish some activity of the software development process? This need for exploration shares some similarities with needs from other domains, such as map exploration, where the information is hierarchically organized, so that zoom and filter mechanisms may be used for seamless navigation through several information abstraction levels.

We aim at providing interactive mechanisms to allow users looking for information pieces they intend to analyze. To achieve this goal, we propose three views focusing on exploratory tasks: *big picture*, *syntax-based view* and *concern-based view*. These views are conceptually abstract, and so can be used with various types of models. Here, we chose *i**, a social goal modeling requirements language [7], to illustrate them. As we will see later, *i** models get complex very quickly. So, it is a good target to illustrate the value of our proposal.

This paper is structured as follows. Section 2 presents an overview of software exploration and *i**. Section 3 defines the three views, illustrates them with *i**, and shows an implementation of the concern-based view. It then discusses how to apply the views to other languages and indicates some challenges on using multiple views for requirements exploration. Section 4 discusses related work and Sect. 5 summarizes our conclusions and discusses ideas for further research.

2 Background

Requirements exploration is a process to navigate through requirements artifacts, aiming at comprehending their structure and content. Each stakeholder engaged in requirements exploration has particular skills and goals, and aims to quickly find specific information to confirm or refute his understanding of the requirements. These skills, goals and understanding can evolve over time. In fact, the faster stakeholders understand artifacts, the faster they may adjust their exploration goals [8]. Similarly to program exploration [9], there are three major reasons to provide mechanisms for requirements exploration:

- Requirements artifacts are often used by people that have not created them. Consequently, these artifacts have unknown structure and content to them.
- Stakeholders need to search information on these artifacts, aiming at completing a software development task or at understanding a domain. Therefore questions may vary from simple (*"Who are the stakeholders?"*) to complex (*"How to modularize the system?"*).
- Requirements are potentially huge, typically written in natural language in several abstraction levels, and scattered among different artifacts which may be specified in distinct languages. Exploration mechanisms can help navigating through the entire documentation to find the elements associated with a specific point of interest.

To illustrate our need for exploration mechanisms, we use the *i** framework. *i** is a goal-oriented requirements framework, whose objective is to analyze and represent how actors collaborate to achieve system goals [7]. *i** offers two models: the Strategic Dependency model (SD), focused on the collaboration among actors, and the Strategic Rationale model (SR), focused on identifying how these goals are achieved by detailing them into tasks.

Usually, modeling starts by building the SD model, showing actors, their main goals and dependencies. After that, in the SR model, each actor is detailed to operationalize the defined goals. Therefore, we may understand the SR model as an expansion of the information offered by the SD model—*i** defines the *boundary* element to group all the elements relevant to an actor. Despite these two levels of abstraction, *i** models may still be considered complex, since its meta-model defines many kinds of elements and relationships. Even using only a subset of these kinds of elements, the model can easily become too large and complex.

Some works have investigated ways to make the *i** notation simpler, for example by increasing its semantic transparency [10]. Instead, we aim at providing visualization mechanisms that essentially use the standard notation, enriched with information hiding mechanisms, so that a particular view supports an easier way to focus on the relevant parts of the model, for a particular stakeholder.

Figure 1 presents an SR model for the Health Care System (HCS), modeled with the iStarLab tool [11]. HCS provides costs management of a medical service, considering the trade-offs between Patients, Insurance Companies and Physicians [7]. This model consists of 13 actors, 13 goals, 41 tasks, 26 softgoals, and 165 relationships. If a stakeholder needs to analyze just one of the actors and its internal activity, the boundary element provides this information, and we could cut out all the other external dependencies. However, if we do that, we lose the context of those elements. On the other hand, if we do not cut them out, we need to manage a larger than actually needed model, and struggle to follow the intricate links. This example shows how the size and complexity of *i** models can be significant, and consequently decrease our ability to analyze them.

3 Views for Requirements Exploration

Creating multiple views is a strategy for requirements exploration. These exploration views must include interaction. *Zoom*, *filter*, *extract*, *details on demand*, *history*, *relate*, and *overview* are examples of exploration tasks supporting interaction with the information [1]. This paper defines three views focusing on filtering and zooming of requirements models. We have taken the classical Visual Information Seeking Mantra, *"Overview first, zoom and filter, then details-on-demand"* [1], into consideration. Thus, the focus is on our need for: top-down and bottom-up navigation; selecting the parts to be detailed; and selecting information types offered for the models, as well as information related to the domain vocabulary.

Our views offer an information subset (raw or pre-processed) of a model taken as source, and represent it by using the same (or similar) notation of the

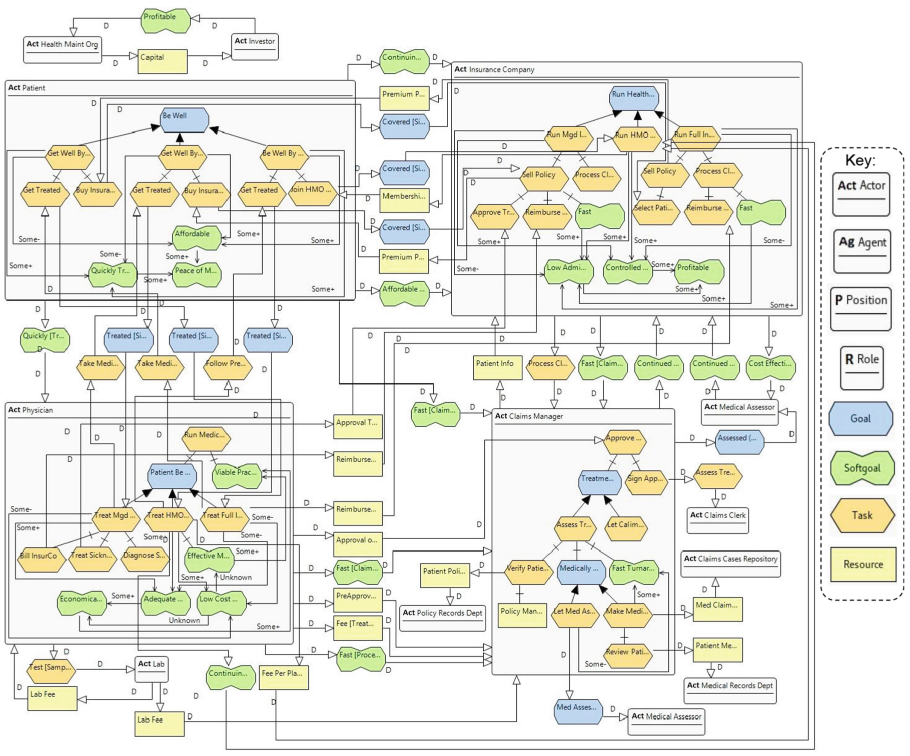

Fig. 1. The HCS strategy dependency model, taken from [11]

source model. Using the representation of the original model may decrease the cognitive effort required from stakeholders. However, this representation should depend on the task that stakeholders are performing and thus other notations may be more appropriated in some situations.

The three views defined in this paper are: the *big picture view*, the *syntax-based view*, and the *concern-based view*. The big picture generates an overview [12] for a source model (or artifact), offering the ability to expand and reduce the details on demand; it organizes the model information on levels of importance or by aggregation. The syntax-oriented view filters the types of language elements that will be visualized. Finally, the concern-oriented view filters concerns, through meta-data, system lexicon (key words) or semantic similarity.

While the big picture generalizes the need of top-down and bottom-up navigation through the information aggregation/disaggregation, the concern and syntax-based views generalize the search for information according to its abstraction level, from instance level to meta-model level, respectively. These views aim to reduce the scope by hiding model elements that are not of interest in a given moment. Next we present a conceptual model of our views.

3.1 Conceptual Model

Figure 2 presents a conceptual model relating our views, the exploration task types and the kind of data that we have taken into consideration. The *visualization* entity consists of *views* and *interaction techniques*. *Views* are projections of a *model* taken as source. A *model* consists of *elements* (*components* and *relationships*), which are characterized by *attributes*. This *model* entity generalizes model approaches (e.g. *i**, use cases, and KAOS). For example, for *i** the set of components includes nodes such as actors, boundaries, goals, softgoals, tasks, and beliefs, while the set of relationships includes dependencies, decomposition, associations, and means-end links.

The *interaction techniques* that we have focused on this paper are *zoom* and *filter*. *Zoom* realizes the *big picture* while *filter* realizes the *syntax* and *concern-based views*. We define the *big picture view* to offer an overview that aggregates *elements* of a model, while the *concern* and *syntax-based views* filter, respectively, data into *attributes* and from the model *elements*.

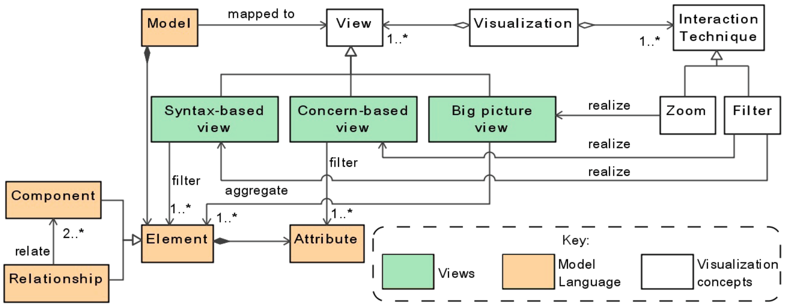

Fig. 2. Conceptual model for exploration views

Although we have classified our views as a kind of zoom and filter, they also include characteristics of the following types of interaction:

– *extract*, because they are slices of an original model, hence representing an information subset of the original model and obeying its syntactic and semantic rules;
– *overview*, because the aggregation proposed in the big picture may generate a collapsed representation at a higher abstraction level;
– *details on demand*, because the aggregations may be gradually expanded according to the user needs.

These views should be used in an interactive and integrative way. This means that it is necessary to provide tools to make the views interactive so that the stakeholder can directly manipulate, as well as combine, them in a non predefined order.

3.2 Exploring *i**

To instantiate our views for *i**, we analyze *i** elements, hierarchy, composition and decomposition characteristics, and intrinsic goal. We model its components and identify which one of them could be filtered and aggregated.

The Big Picture View. We can perceive the SD model as an abstract view of a system, showing its context. The SD model is later refined into SR models, each specifying an actor in the SD model. Here, the SD model can be seen as a Big Picture for the SR models.

SD models usually show more than one dependency between any two actors. The larger the number of links (*i.e.*, the actor's fan-in and fan-out), the more unreadable the model becomes. Hence, an SD overview, or simpler view, may be useful, allowing us to gradually expand and reduce it according to our needs. It is worth mentioning that we want to create ways to aggregate (and disaggregate) information into models, rather than creating new notations to represent them. So, if the source model language does not provide aggregation elements, we must create visual mechanisms to help the user see the collapsed points of information.

The intrinsic characteristics of SD and SR models led to us to identify four ways to reduce their quantity of elements:

- Hiding dependencies: multiple dependencies between any two actors are collapsed into one single relationship. This relationship will be annotated in both directions, defining the quantity of dependencies collapsed into it.
- Hiding actors: *actors* (as well as *roles*, *positions* and *agents*) associated to others by *isa*, *is-part-of*, *plays*, *occupies* or *ins* relationships may be omitted since they do not have dependencies with other actors. For example, if actor1 *isa* actor2, then actor1 may be collapsed.
- Hiding the internal elements of an actor: the boundary element may be used to reduce the size of the model by hiding its internal elements. This is an *i** resource that tools have already explored.
- Hiding relationships among tasks, goals, softgoals and beliefs: any two elements only linked by decomposition relationships are collapsed into the most general element. Note that means-ends and contributions are not hidden, as they are not hierarchical relationships.

Figure 3 depicts the visual mechanisms our approach adds to *i** models, so that the stakeholders are made aware of available model exploration alternatives at a given moment. The plus sign in an actor denotes the possibility of expanding the information on that actor, while the minus sign provides the alternative for collapsing (*i.e.*) that information. Finally, a simple line aggregates several dependencies that can, if necessary, be expanded.

Figure 4 illustrates our big picture view for Fig. 1. It is a simplification of the original model, since it hides several visual elements. This reduction still shows the actors and dependencies among them. Note that dependencies are represented by an annotated relationship, whose ends indicate how many dependencies exist on each direction, if any. Although it uses a different notation for

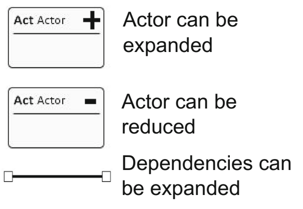

Fig. 3. Visual clues for simplifying *i** models visualization

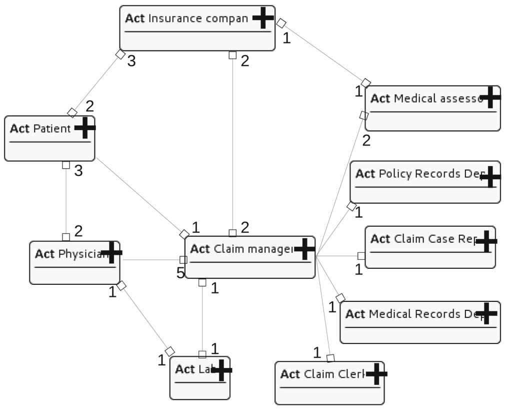

Fig. 4. The Big Picture with both actors and dependencies collpased

dependencies and actors, there are no new elements involved, hence still representing a subset of an SD model. The number of dependencies collapsed in each relation may be used to identify, for example, actors that accumulate too many responsibilities, or those that have too few (their fan-in and fan-out).

Stakeholders may explore (collapse and expand) actors and relationships (their elements of interest), by directly selecting them or by using other interaction elements such as menus and check boxes. For example, Fig. 5 illustrates the expansion of *Patient* actor and dependencies between *Physician* and *Insurance Company* actors, which were collapsed in Fig. 4.

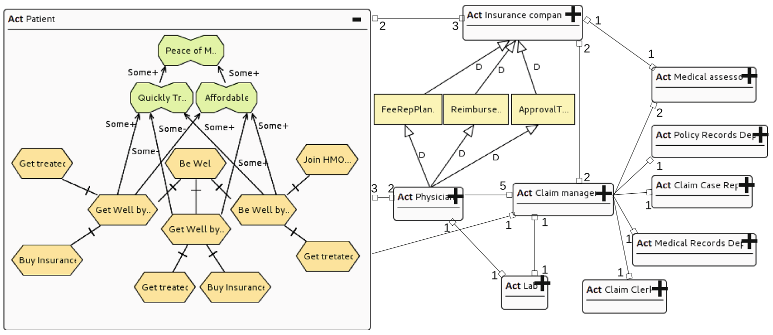

Fig. 5. The Big Picture with an expanded link and actor

Syntax-Based View. This view provides the possibility of choosing the syntactic elements of interest. It is worth noting that hiding some combinations of elements (subsets of the language) may generate semantically invalid models (e.g., showing the boundary but not its related actor). Therefore, it is necessary to define the possible combinations of elements through rules to generate well-formed views. Another alternative is to use visual clues to contrast *focused* with *non-focused* elements.

Figure 6 depicts a SD model showing only the resource dependencies. This view helps stakeholders to identify the resources flow among actors, for example, *Lab* can receive a *Lab fee* from *Claim manager* and *Physician*. Other important filters are related to each dependum, node or relationship kinds. For instance, generating a view that shows only the goals (softgoals, tasks or beliefs) related to each actor helps to understand why those actors depend on each other, or if their goals are not directly related to their dependencies.

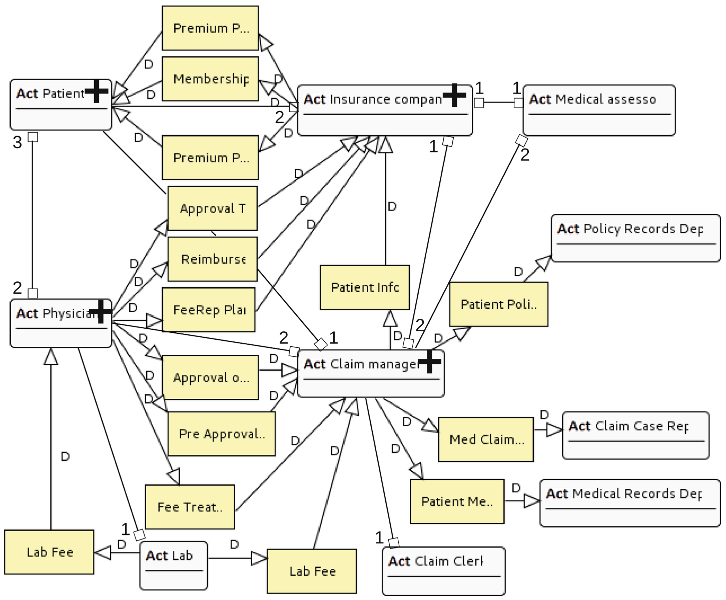

Fig. 6. Syntax-based view: filtering resource dependums

Concern-Based View. This view allows the abstraction of the model by filtering model elements. The focus is not on a type of element of the language, but, instead, on values for these types. For instance, if stakeholders only need to explore one specific actor through its SR model or identify which aspects concerning response time have been addressed, the actor's name and keywords about response time, respectively, may be used as criteria for generating these views. Additionally, we can use the syntax-based view in conjunction with the concern-based view to make the filter more precise. For example, if the name of that actor is being used for naming another kind of element, we could narrow the search by only considering the actor element type.

This kind of visualization also represents a set of views that could be defined by the stakeholders. Basically, they can specify values for particular types of language elements (in this case the syntax and concern based views are used in conjunction), or, more freely, search for a concern on any type of element. It is also necessary to consider that the result of these searches has to include the context (the model elements) related to the concern searched, as this is usually relevant for the analysis. For instance, when stakeholders search for the actor named *Patient*, they probably want actor *Patient* with all its internal elements (those within its boundary), and the actors with dependencies to *Patient*.

In this case, there is a clear need to consider the distance to the elements to be captured by this view. For example, a distance of zero only captures the elements directly searched, a distance of one considers the elements directly searched and those elements linked to them, a distance of two searches all elements in distance of one and those elements linked to them, and so on and so forth. Moreover, some kind of query language could be needed.

Figure 7 illustrates the result for a free search, requiring the elements that match the value *Cost*, with a distance of one. The result is all elements with the string *Cost* and the actors, tasks, goals and softgoals directly linked to them.

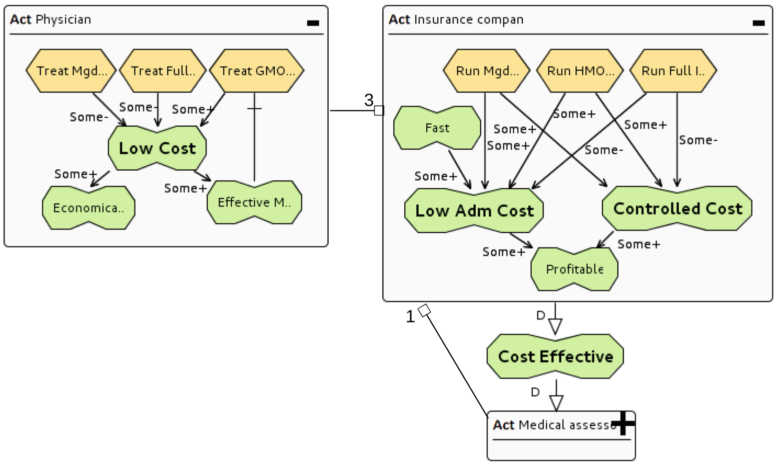

Fig. 7. Concern-based view (considering a distance of one): filtering the string *Cost*

3.3 Implementation

The *i** models presented in this paper were created with our iStarLab tool, an Eclipse-based *i** editor that allows stakeholders to generate *concern-based views* by selecting one or more concerns. After the selection, the tool presents a view that shows which elements of the model are involved or contribute to achieve that concern or set of concerns. The prototype tool was implemented using Domain Specific Language (DSL) construction mechanisms.

The tool can be used to create *i** models as well as to analyze them in terms of concerns. In this context, a concern can be viewed, for example, as a non-functional requirement (NFR) or a symbol of the system lexicon. Each one of the model elements should have one or more concerns associated with it, through tags. The stakeholder can assign one or more concerns to the elements at any time. During or after the modeling process, s/he can choose a set of those concerns, from a list, for further scrutiny. With this analysis, stakeholders can, for example, perceive if there are elements that should have associated a specific concern, which are the most used concerns, and which resources are involved or are needed to achieve a desired concern.

In this implementation of the concern-based view, there are two types of visualizations available: *(i)* highlight model elements with a specific set of concerns, without losing the model context; and *(ii)* view only model elements with a specific set of concerns, i.e., the others should be hidden in the model. After analyzing a given concern, the stakeholder can view the model in its original state, i.e., the model without highlighted elements or deleted ones.

3.4 Discussion and Challenges

Big picture, syntax and concern-base views were defined in an abstract manner. So, they can be instantiated to other types of requirements models. To instantiate these views, it is necessary to analyze the model principles and their syntactic and semantic elements and structure. Also, it is necessary to analyze if the model has aggregation elements that may be used, similarly to the *boundary* element in *i**. To validate this claim about the generality of these views, we successfully applied them to use case diagrams and their scenario descriptions (the results cannot be presented here, due to lack of space).

Taking into consideration the performed literature review and our experience with the *i** and use case models, we enumerate some of the challenges on generating or using multiple views for requirements exploration: *(i)* in the requirements engineering process, many different models are generated; so, it is necessary to provide exploration mechanisms to navigate through them, instead of only navigating an isolated model; *(ii)* the proposed views complement each other, so it is necessary to define, for each model, how they may be composed; *(iii)* the tools implementing these views should provide mechanisms to allow users to interact directly with the visual elements; *(iv)* this interaction includes generating other views from the resulting views, and so, care must be taken to avoid confusion between the source model and its views; *(v)* since users may interact with the source model as well as with the views, it is also necessary to generate a view about the path followed to the achieved result; *(vi)* interaction mechanisms include aspects from human-computer interaction have not been taken into consideration yet.

4 Related Work

For software comprehension, it is necessary to enhance bottom-up and top-down comprehension approaches, facilitate the navigation, provide orientation clues, and reduce disorientation [9]. Tools that support program comprehension should provide requirements such as browse, search and filter mechanisms, as well as abstractions, history and multiples views [13,14]. Actually, software exploration requires flexible and interactive views, i.e., visualization techniques rather than static and isolated views, in order to enable the user navigating through artifacts [14–18].

Visualization includes data types and interaction techniques [1,8,19]. In this paper, we have considered that our data types are basically software, graphs and text, while the task types are filter and zoom. Therefore, there is a wide set of other alternatives that may also be used for requirements comprehension.

Views generated for software comprehension serve firstly to reveal and understand software structures and behavior. Consistency is secondary, as these views are generated when needed and may be discarded soon after, to be regenerated when necessary. This idea comes from the information visualization field, where visualization is an activity rather than an artifact [17]. There is a high potential for this type of visualization in requirements engineering, due to the emphasis on information seeking and creation, with multiple parties and activities involved [16,17].

In this context, we may separate the related works into two categories: those that propose views generated from data sets (that describe requirements and their attributes), and those that propose views generated from constructs (or properties) of a meta-model. In the first category, filters and attributes of requirements are used to generate graphical views, usually graphs or charts [4,6,16,20]. Although they have inspired our work, they are distinct because neither do they deal with the properties of a modeling language nor with the generation of visual clues in the source model (or source model subset).

In the second category (see Table 1) we list approaches that generate views from models like $i*$ [5,21], theme/doc [22], use cases [23] and NFR graphs [24]. This category is highly related to our work. In general, these views are distinct from the views defined in our research, because: *(i)* they deal with a specific kind of input, while ours are abstract enough to be applied to many kinds of models; *(ii)* they generate static views by using a very specific criterion, while ours use criteria defined by users, so that many views can be generated; or *(iii)* they support only one way of interacting, while in our approach, views abstract three ways to interact with requirements models.

Horkoff and Yu [5] present two views (or filters): one to highlight the starting points for analysis (the leaves of model), and another to indicate the elements involved in a conflict. These views focus on seeking for elements of the meta-meta model that match with pre-defined properties, while our views are generic to accommodate user-defined properties.

Ernst, Yu and Mylopoulos [21] propose a visualization scheme where quality attributes are added to elements of $i*$ models to enable the projection of views

Table 1. Requirements aggregation and filtering mechanisms

Ref.	Source	Target	Static/Interactive	Pre-defined criteria
[21]	*i**, meta-data	*i**	Interactive	Concern-based
[5]	*i**	*i**	Static	Syntax-based: leaves and elements in conflict
[22]	ThemeDoc	ThemeDoc	Interactive	Aggregation
[23, 25]	Use cases	Use cases, UML	Static	Aggregation and concern-oriented
[24]	NFR frame-work	NFR frame-work	Static	Syntax-based: objective, problem, alternative and selection patterns
Our views	*i**, use cases, others	*i**, use cases, others	Interactive	Aggregation, concern and syntax-based

based on these attributes. In their work, the quality attributes of efficiency, trustability, certainty and feasibility are defined and showed on the goal model by using visual clues. The difference between this work and ours is that it does not provide other filters types, neither overview.

Baniassad and Clarke [22] define a summarized view of Theme/Doc to deal with the lack of scalability of this kind of model. This view is similar to our Big Picture, and it was idealized to have the same kind of interaction we claim it is needed. However, their view has not been designed to be generalized nor to support filters.

Jacobson and Ng [25] define an approach where use case slices are elaborated, including pieces of the use case model and other UML models that deal with a specific concern. We can understand this use case slice as a static concern-oriented overview of the UML models, but it is not automatically (or semi-automatically) generated. Furthermore, Jacobson, Spence and Bittner [23] reinforce the importance of a Big Picture, but in this case, it is the use case diagram (unchanged). In opposition to that, our approach considers that an integrated model exists and from it a tool should be able to generate the slices, and these slices are user-defined, by considering three abstract criteria.

Supakkul and Chung [24] present a framework for visualization of patterns (objective, problem, alternatives and selection patterns) in NFRs graphs. Therefore, the criterion for visualization is specific for these patterns, so that users cannot make free searches.

5 Conclusions and Future Work

This work presents three views for models exploration: *big picture view*, *syntax-based view* and *concern-based view*. These views are based on the interaction tasks zoom and filter. Therefore, they capture three manners of abstracting a model, by decreasing its amount of elements, making it possible for stakeholders to search and focus on information of interest. Although only an instantiation of these views were shown for *i** models, they are abstract enough to be applied to other kinds of models and we have done so already for use case models.

The related works diverge from our abstract views because they are well tailored to specific languages or concerns. Instead, our views are to be adapted to different languages, and capture many kinds of concerns. Therefore, our proposal provides a strategy to effectively deal with the complexity of requirements models, where our views offer more flexible mechanisms for exploring and understanding such models. Without these kinds of mechanisms, more stakeholders' effort is demanded to find and analyze relevant information in the system model.

For the near future, we are interested in investigating how tools can be prepared for supporting our views. We are already exploring the use of DSLs to query requirements models, and meta-data to provide richer insights. Also, we will focus on other interaction tasks and define the variabilities that are intrinsic to requirements exploration, visualization and comprehension, as well as to define a process to instantiate our views to other requirement models. We plan to conduct experimental evaluations of the impact of introducing the proposed mechanisms in requirements tools and on the efficiency and effectiveness of different stakeholders while performing requirements exploration. Approaches to manage consistency among models such as in [26] will be also considered.

Acknowledgments. This work was funded by UFRN and NOVA LINCS research laboratory (Ref. UID/CEC/04516/2013), CNPq-PDE grant 201848/2014-7, and FCT-MCTES research grant SFRH/BD/108492/2015.

References

1. Shneiderman, B.: The eyes have it: a task by data type taxonomy for information visualizations. In: Symposium on Visual Languages, pp. 336–343. IEEE (1996)
2. Diehl, S.: Software Visualization: Visualizing the Structure, Behaviour, and Evolution of Software. Springer Science & Business Media, New York (2007)
3. Cooper Jr., J.R., Lee, S.W., Gandhi, R., Gotel, O.: Requirements engineering visualization: a survey on the state-of-the-art. In: 4th International Workshop on Requirements Engineering Visualization (REV 2009), pp. 46–55. IEEE (2009)
4. Donzelli, P., Hirschbach, D., Basili, V.: Using visualization to understand dependability: a tool support for requirements analysis. In: 29th Annual IEEE/NASA Software Engineering Workshop, pp. 315–324. IEEE (2005)
5. Horkoff, J., Yu, E.: Visualizations to support interactive goal model analysis. In: 5th International Workshop on Requirements Engineering Visualization (REV 2010), pp. 1–10. IEEE (2010)

6. Reddivari, S., Rad, S., Bhowmik, T., Cain, N., Niu, N.: Visual requirements analytics: a framework and case study. Requirements Eng. **19**(3), 257–279 (2014)
7. Yu, E.: Modelling strategic relationships for process reengineering. Ph.D. thesis, University of Toronto, Canada (1996)
8. Keim, D.: Information visualization and visual data mining. IEEE Trans. Visual Comput. Graphics **8**(1), 1–8 (2002)
9. Storey, M.A.D., Fracchia, F.D., Müller, H.A.: Cognitive design elements to support the construction of a mental model during software exploration. J. Syst. Softw. **44**(3), 171–185 (1999)
10. Moody, D., Heymans, P., Matulevičius, R.: Visual syntax does matter: improving the cognitive effectiveness of the i* visual notation. Requirements Eng. **15**(2), 141–175 (2010)
11. Gralha, C., Goulão, M., Araújo, J.: Identifying modularity improvement opportunities in goal-oriented requirements models. In: Jarke, M., Mylopoulos, J., Quix, C., Rolland, C., Manolopoulos, Y., Mouratidis, H., Horkoff, J. (eds.) CAiSE 2014. LNCS, vol. 8484, pp. 91–104. Springer, Heidelberg (2014). doi:10.1007/978-3-319-07881-6_7
12. Hornbæk, K., Hertzum, M.: The notion of overview in information visualization. Int. J. Hum. Comput. Stud. **69**(7), 509–525 (2011)
13. Kienle, H.M., Müller, H., et al.: Requirements of software visualization tools: a literature survey. In: 4th IEEE International Workshop on Visualizing Software for Understanding and Analysis, (VISSOFT 2007), pp. 2–9. IEEE (2007)
14. Storey, M.A.D.: Theories, methods and tools in program comprehension: past, present and future. In: 13th International Workshop on Program Comprehension (IWPC 2005), pp. 181–191. IEEE (2005)
15. Favre, J.M.: A new approach to software exploration: back-packing with GSEE. In: 6th European Conference on Software Maintenance and Reengineering, pp. 251–262. IEEE (2002)
16. Gotel, O., Marchese, F.T., Morris, S.J.: On requirements visualization. In: 2nd International Workshop on Requirements Engineering Visualization (REV 2007). IEEE (2007)
17. Gotel, O., Marchese, F.T., Morris, S.J.: The potential for synergy between information visualization and software engineering visualization. In: 12th International Conference on Information Visualisation, (IV 2008), pp. 547–552. IEEE (2008)
18. Niu, N., Mahmoud, A., Yang, X.: Faceted navigation for software exploration. In: IEEE International Conference on Program Comprehension, pp. 193–196 (2011)
19. Keller, P.R., Keller, M.M.: Visual Cues: Practical Data Visualization. IEEE Computer Society Press, Los Alamitos (1994)
20. Heim, P., Lohmann, S., Lauenroth, K., Ziegler, J.: Graph-based visualization of requirements relationships. In: 3rd International Workshop on Requirements Engineering Visualization, (REV 2008), pp. 51–55. IEEE (2008)
21. Ernst, N., Yu, Y., Mylopoulos, J.: Visualizing non-functional requirements. In: 1st International Workshop on Requirements Engineering Visualization (REV 2006). IEEE (2006)
22. Baniassad, E., Clarke, S.: Investigating the use of clues for scaling document-level concern graphs. In: Workshop on Early Aspects (held with ECOOP 2004), Vancouver, Canada, pp. 1–7 (2004)
23. Jacobson, I., Spence, I., Bittner, K.: Use Case 2.0: the guide to succeeding with use cases. In: Ivar Jacobson International, pp. 1–55 (2011)

24. Supakkul, S., Chung, L.: Visualizing non-functional requirements patterns. In: 5th International Workshop on Requirements Engineering Visualization (REV 2010), pp. 25–34. IEEE (2010)
25. Jacobson, I., Ng, P.W.: Aspect-Oriented Software Development with Use Cases. Addison-Wesley Object Technology Series. Addison-Wesley Professional, Reading (2004)
26. Bork, D., Buchmann, R., Karagiannis, D.: Preserving multi-view consistency in diagrammatic knowledge representation. In: Zhang, S., Wirsing, M., Zhang, Z. (eds.) KSEM 2015. LNCS, vol. 9403, pp. 177–182. Springer, Heidelberg (2015). doi:10.1007/978-3-319-25159-2_16